

# A Local Logic for Realizability in Web Service Choreographies

R. Ramanujam

IMSc, Chennai

jam@imsc.res.in

S. Sheerazuddin

SSNCE, Chennai

sheerazuddins@ssn.edu.in

Web service choreographies specify conditions on observable interactions among the services. An important question in this regard is realizability: given a choreography  $C$ , does there exist a set of service implementations  $I$  that conform to  $C$ ? Further, if  $C$  is realizable, is there an algorithm to construct implementations in  $I$ ? We propose a local temporal logic in which choreographies can be specified, and for specifications in the logic, we solve the realizability problem by constructing service implementations (when they exist) as communicating automata. These are nondeterministic finite state automata with a coupling relation. We also report on an implementation of the realizability algorithm and discuss experimental results.

## 1 Introduction

The study of composition of distributed web services has received great attention. When we know what kind of services are available, specifying a sequence of communications to and from them can well suffice to describe the overall service required. Such a *global* specification of interaction composition has been termed **choreography** ([6]). The distributed services can then be synthesized as autonomous agents that interact in conformance with the given choreography. This offers an abstract methodology for the design and development of web services. The choreography and its implementation may be put together in a **choreography model** [26],  $M = (C, I)$ , where, as already mentioned,  $C$  is a specification of the desired global behaviours (a choreography), and  $I$  a representation of local services and their local behaviours (an implementation) which collectively should satisfy the specified global behaviour. A **choreography modeling language** [26] provides the means to define choreography models, i.e., choreographies, service implementations, and their semantics including a mechanism to compare global behaviors generated by service implementations with a choreography.

A principal challenge for such a methodology is that choreographies be **realizable** ([16]). What may seem simple global specifications may yet be hard, or even impossible, to implement as a composition of distributed services. The reason is simple: while the global specification requires a communication between 1 and 2 to precede that between 3 and 4, the latter, lacking knowledge of the former, may well communicate earlier. Thus the composition would admit forbidden behaviours. In general, many seemingly innocuous specifications may be unrealizable. Even checking whether a choreography is realizable may be hard, depending on the expressive power of the formalism in which the choreography is specified.

Closely related, but more manageable, is the problem of **conformance**: check whether a given set of services implement the given choreography specification. Once again, the expressive power of the specification formalism is critical for providing algorithmic solutions to the problem.

The two problems relate to the satisfiability and model checking problems of associated logics. Since the 1980's a rich body of literature has been built in the study of such problems ([9]).

In the literature, choreographies have been formally specified using automata [13], UML collaboration diagrams [5], interaction Petri nets [10], or process algebra [8]. The service implementations have been modelled variously as Mealy machines, Petri nets or process algebra. Visual formalisms (such as message sequence charts [24]) are naturally attractive and intuitive for choreography specifications but can be imprecise. For instance, it is hard to distinguish between interactions that are permissible and those that must indeed take place. While machine models are precise they might require too much detail.

A natural idea in this context is the use of a logical formalism for choreography specification and that of finite state machines for their implementation. When the formulas of the logic specify global interaction behaviour and models for the logic are defined using products of machines, realizability and conformance naturally correspond to the satisfiability and model checking problems for the logic.

Once again, a natural candidate for such a logic is that of temporal logic, linear time or branching time ([22], [9]). One difficulty with the use of temporal logics for global interaction specifications is that sequentiality is natural in such logics but concurrency poses challenges. It is rather easy to come up with specifications in temporal logics that are not realizable. On the other hand, if we wish to algorithmically decide whether a temporal specification is realizable or not, the problem is often undecidable, and in some cases of high complexity even when decidable. (See [2] and [1] for decidability of the closely related problem of realizability of message sequence graphs.)

One simple way out is to design the temporal logic, limit its expressiveness drastically, so that we can ensure *by diktat* that every satisfiable formula in it is realizable. This is the line we follow here, initiated by [27] and developed by [23], [21]. In such **local** temporal logics, we can ensure realizability by design. The models for these logics are presented as a system of communicating automata (SCA). Both realizability and conformance are decidable in this setting.

Our work is similar to that of McNeile [19] who extend the process algebra based formalism of Protocol Modeling [20] to define a notion of protocol contract and describe choreographies and participant contracts. They give sufficient conditions for realizability in both synchronous and asynchronous collaborations.

The language-based choreography realizability problem considered in this paper was proposed for conversation protocols in [13] where sufficient conditions for realizability were given. Halle & Bultan [14] consider the realizability of a particular class of choreographies called arbitrary-initiator protocols for which sufficiency conditions given in [13] fail. The algorithm for choreography realizability works by computing a finite-state model that keeps track of the information about the global state of a conversation protocol that each peer can deduce from the messages it sends and receives. Thereafter, the realizability can be checked by searching for disagreements between peers' deduced states.

[25] model choreography as UML collaboration diagrams and check their realizability. They have also implemented a tool which not only checks the realizability of choreography specified using collaboration diagrams but also synthesize the service implementations that realize the choreography [4].

[3] consider the realizability problem for choreographies modelled as conversation protocols [13] (finite automata over send events). They give necessary and sufficient conditions which need to be satisfied by the conversations for them to be realizable. They implement the proposed realizability check and show that it can efficiently determine the realizability of a subclass of contracts [11] and UML collaboration diagrams [5], apart from conversation protocols.

The work on session types [15] is also related to realizability of conversation protocols and has been used as a formal basis for modelling choreography languages [8].

The work presented in [18] checks choreography realizability using the concept of controllability. Given a choreography description, a monitor service is computed from that choreography. The monitor service is used as a centralized orchestrator of the interaction to compute the distributed peers. The

choreography is said to be realizable if the monitor service is controllable, that is, there exists a set of peers such that the composition of the monitor service and those peers is deadlock-free.

Pistore et al., [17] present a formal framework for the definition of both global choreography as well as local peer implementations. They introduce a hierarchy of realizability notions that allows for capturing various properties of the global specifications, and associate specific communication models to each of them. Finally, they present an approach, based on the analysis of communication models, to associate a particular level of realizability to the choreography.

Decker and Weske [10] model choreography as interaction Petri nets, an extension of Petri nets for interaction modelling, and propose an algorithm for deriving corresponding behavioural interfaces. The message exchanges are assumed to be asynchronous in nature. They define two properties, realizability and local enforceability, for interaction Petri nets and introduce algorithms for checking these properties.

In the light of this literature, the need and relevance of the work presented in this paper is naturally questionable. The contribution of this paper is two fold: one, to argue that **partial orders** provide a natural way of describing potential interactions – sequential and concurrent; two, to suggest that a test for realizability be translated to an expressiveness restriction on the formalism for choreography specification in such a way that *every* consistent specification is realizable. While realizability by design is not in itself new, the application of automata based methods on partial orders can lead to new ways of defining ‘good’ choreographies. A major advantage of such partial order based specification is that we can reason about components (services) separately, and limit global reasoning to the minimum required. In terms of worst case complexity this makes no difference, but in practice this is of great use.

The paper is organized as follows. In the next section, we define choreography realizability and discuss examples of realizable and unrealizable choreographies. Further, we note that even though choreographies, modelled as conversation protocols, are defined over send events they give rise to partial order behaviours in service implementations. We then propose  $p$ -LTL, which admits only realizable choreographies, and present systems of communicating automata (SCA) to model sets of service implementations admitted by choreographies in  $p$ -LTL. We show that realizability is decidable, and discuss some experiments in implementing the decision algorithm. Detailed proofs are relegated to the Appendix.

## 2 Choreography realizability

A choreography specification  $C$  is **realizable** if there is an **implementation**  $I$  of the interacting services such that once the system is initialized, its processes behave according to the choreography specification.

Consider the choreography  $C_0$  [7] represented as conversation protocol, given in Figure 1. The conversation protocol, a collection of sequences of send events (conversations), is modelled as a nondeterministic finite state automaton. Let  $L(C_0)$  denote the set of all conversations in  $C_0$ . In  $C_0$  there are three services interacting with each other: John ( $j$ ), Agent ( $a$ ) and Hotel ( $h$ ). John wants to take a vacation. He has certain constraints about where he wants to vacation, so he sends a query to his Agent stating his constraints and asking for advice. The Agent responds to Johns query by sending him a suggestion. If John is not happy with the Agents suggestion he sends another query requesting another suggestion. Eventually, John makes up his mind and sends a reservation request to the hotel he picks. The hotel responds to Johns reservation request with a confirmation message.

One set of service implementations  $I_0$  for the choreography  $C_0$  is given in Figure 2. Each service is implemented as a nondeterministic finite state automaton (NFA) over send and receive events, equipped with FIFO queues for sending message to other services. The causal dependence among these events (e.g. that a message can be received only after it is sent) is represented by couplings, shown as  $\lambda$  labelled

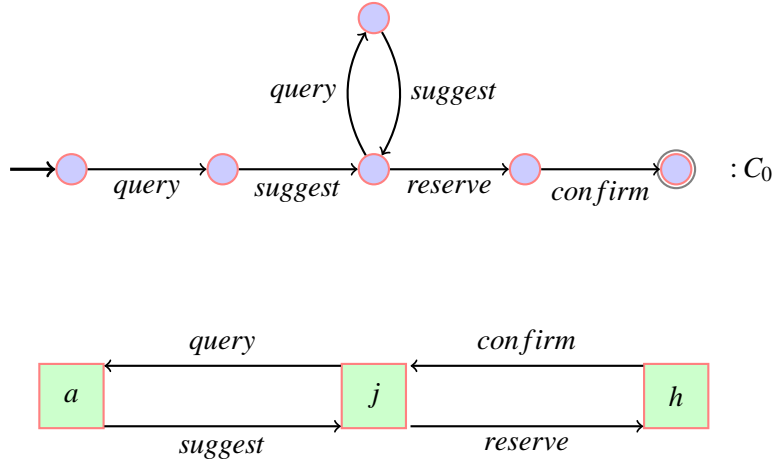


Figure 1: A Realizable Choreography

arrows. For example, consider the case when service  $s_2$  sends a message *query* to service  $s_1$ . The transition  $(q_0, ?query, q_1)$  in  $I_0$  must be coupled with  $(q_4, !query, q_5)$ , that is,  $(q_0, ?query, q_1)$  must happen before  $(q_4, !query, q_5)$ . This coupling is represented by a  $\lambda$  labelled arrow from the target state of  $!query$  event ( $q_1$ ) to target state of  $?query$  event ( $q_5$ ).

Considering the behaviour specified by  $C_0$  as a collection of sequences of send events is natural and simple, but hides concurrency information. Two send events by distinct services, locally determined by them, can proceed simultaneously or in any order. Therefore, the interactions among service implementations  $I_0$  are better viewed as partial orders on send and receive events. Let  $\mathcal{L}$  be the set of all such partially ordered executions of  $I_0$ . We call the objects in  $\mathcal{L}$  as **diagrams**. A sample of the diagrams in  $\mathcal{L}$  is given in Figure 3.

Let  $D$  be an arbitrary diagram in  $\mathcal{L}$  and  $Lin(D)$  be the set of all linearizations of  $D$ . A linearization of  $D$  is a linear order on the events in  $D$  which respects the given partial order. Let  $Chor(D)$  be obtained from  $Lin(D)$  by removing receive events from each sequence. Let  $Chor(\mathcal{L}) = \bigcup_{D \in \mathcal{L}} Chor(D)$ . We say that  $I_0$  realizes  $C_0$  if  $Chor(\mathcal{L}) = L(C_0)$ . For our example, this is easily seen to be the case.

Even though  $C_0$  turned out to be realizable, we can easily fashion choreographies that are not realizable [6]. Consider the choreography  $C_1$  with  $L(C_1) = \{a \cdot b\}$ , given in Figure 4, where service  $s_0$  sends a message  $a$  to  $s_1$  and  $s_2$  sends  $b$  to  $s_3$ .

Note that there is no causal dependence between a send event  $!a$  and a send event  $!b$  and hence any set of service implementations defined by projection will also admit the global behaviour  $b \cdot a$ . This situation is illustrated in Figure 5.

We can define choreographies where the reasons for non-realizability are not so obvious. Consider the choreography  $C_2$ , with  $L(C_2) = \{a \cdot b \cdot c, b \cdot a\}$ , over three services  $s_4, s_5, s_6$  and messages  $a, b, c$  shown in Figure 4. Since every service has a FIFO queue, it can be shown that every implementation that permits the two sequences in  $L(C_2)$  will also permit the sequence “ $b \cdot a \cdot c$ ” that is not in  $C_2$  [12].

Note that these unrealizable choreographies can be easily specified by formulas of a standard temporal logic such as LTL, the temporal logic of linear time. The following formula specifies the choreography  $C_1$ :

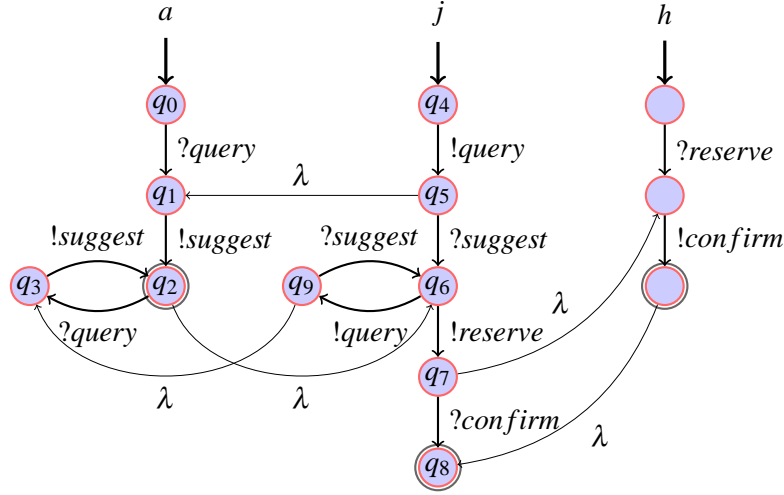


Figure 2: A Sample Service Implementation for Choreography in Figure 1

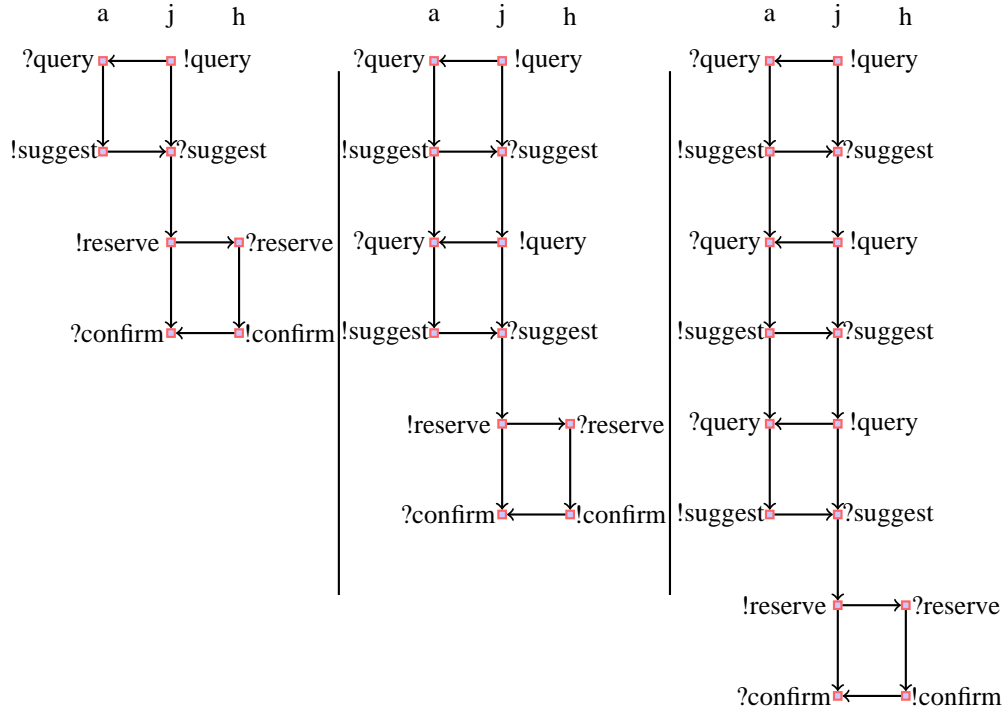


Figure 3: Sample Partial Order Executions of Service Implementations in Figure 2

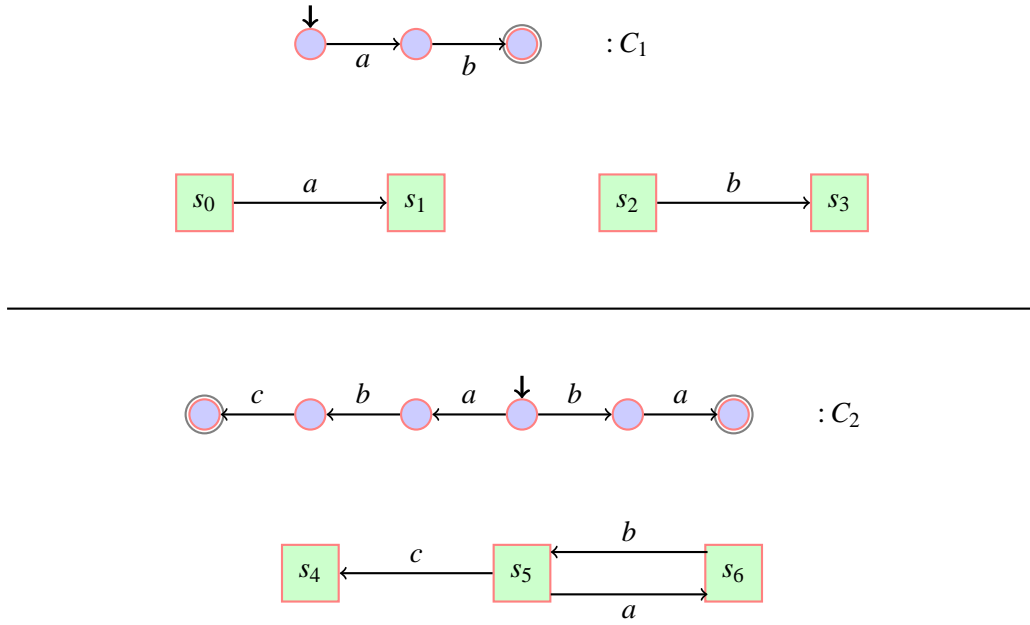
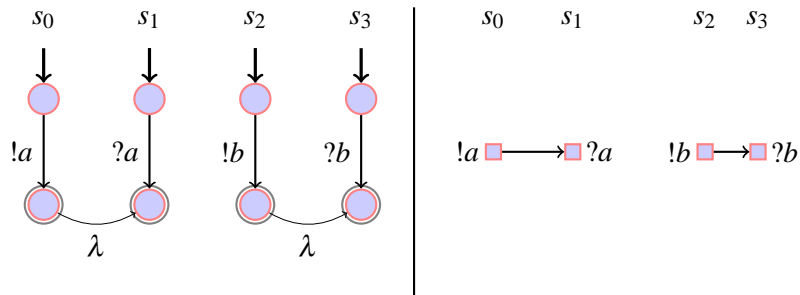


Figure 4: Unrealizable choreographies [26]

Figure 5: Sample Implementation of Choreography  $C_1$  in 4 and its Partial Order Execution

$$\Diamond(!a_0^1 \wedge \Diamond !b_2^3) \wedge \Box(!b_2^3 \supset \Box \neg !a_0^1).$$

Above, we have used the natural encoding  $!a_i^j$  to denote a send event from service  $i$  to service  $j$ . It says that send-to- $s_1$  event in  $s_0$  happens before send-to- $s_4$  event in  $s_3$ . It comes with an extra sanity check: there is no send-to- $s_1$  event in  $s_0$  after send-to- $s_4$  event in  $s_3$ . On sequences, this is a satisfiable formula. But as we have already discussed, such a choreography specification is unrealizable. This forms our motivation for considering a local temporal logic on partial orders where such specifications are unsatisfiable.

### 3 Logic

The logical language which we use to specify choreographies is a **local** temporal logic. It is named as  $p$ -LTL.

#### 3.1 Syntax and Semantics

We fix the set of  $n$  services  $Ag = \{s_1, s_2, \dots, s_n\}$ . Further, we fix countable sets of *propositional letters*  $P_s$ , for local properties of service  $s \in Ag$  and  $\mathcal{M}$  as the countable set of message symbols. The propositional symbols in  $P_s$  are intended to specify internal actions in web service  $s$ . We assume, for convenience, that  $P_s \cap P_{s'} = \emptyset$  for  $s \neq s' \in Ag$ .

The syntax of *s-local formulas*, local service formulas, is given below:

$$\alpha \in \Phi_s ::= !a_{s'}, a \in \mathcal{M} \mid ?a_{s'}, a \in \mathcal{M} \mid p \in P_s \mid \neg \alpha \mid \alpha_1 \vee \alpha_2 \mid \bigcirc \alpha \mid \Diamond \alpha \mid \ominus \alpha$$

$\bigcirc$  is the **next**,  $\Diamond$  is the **eventual** and,  $\ominus$  is the **previous** temporal modality.  $!a_{s'}$  is a send- $a$ -to- $s'$  proposition in  $s$  whereas  $?a_{s'}$  is the corresponding receive- $a$ -from- $s'$  proposition.

*Global formulas* are obtained by boolean combination of local formulas:

$$\psi \in \Psi ::= \alpha @ s, \alpha \in \Phi_s, s \in Ag \mid \neg \psi \mid \psi_1 \vee \psi_2$$

The propositional connectives  $\wedge, \supset, \equiv, \oplus$  and derived temporal modality  $\Box$  are defined as usual. In particular,  $\Box \alpha \equiv \neg \Diamond \neg \alpha$ . Fix  $p_0 \in P_s$  and let  $True = p_0 \vee \neg p_0$ ; let  $False = \neg True$ .

A choreography is a global formula  $\psi \in \Psi$ , that intuitively starts the services off in a global state, and their local dynamics and interactions are given by service formulas. Note that global safety properties can be specified by a conjunction of local safety properties.

The formulas are interpreted on a class of partial orders, defined as follows.

$M = (E_{s_1}, \dots, E_{s_n}, \leq, V)$  such that:

- $E_{s_1}, \dots, E_{s_n}$  are finite nonempty sets of events.  $E_{s_i}$  is the set of events associated with service  $s_i$ . We assume that there is a unique event  $\perp_{s_i} \in E_{s_i}$  for each  $i$ . Let  $E = \bigcup_{i \in [n]} E_{s_i}$ .

- $\leq \subseteq (E \times E)$  is a partial order. Let  $\prec$  be the one-step relation derived from  $\leq$ , that is:  $\leq = \prec^*$ .

Define, for each service  $s_i$ ,  $\leq_{s_i} = \leq \cap (E_{s_i} \times E_{s_i})$ . It gives the local behaviour of  $s_i$ . We require that  $\perp_{s_i}$  be the unique minimum event in  $E_{s_i}$ ; that is, for all  $e \in E_{s_i}$ , we have:  $\perp_{s_i} \leq_{s_i} e$ . Let  $\prec_{s_i}$  be the one-step relation induced by  $\leq_{s_i}$ .

Now define, across services,  $\leq_c \subseteq E \times E$  by:  $\leq_c = \{(e, e') \mid e \prec e', e \in E_{s_i}, e' \in E_{s_j}, i \neq j, e \neq \perp_{s_i}\}$ . It gives the global communication pattern among the services: interpret  $(e, e')$  above as  $s_i$  sending

a message to  $s_j$  with  $e$  being the send event and  $e'$  being the corresponding receive event. Note that the initial event cannot be a communication event.

- $V : E \rightarrow 2^{(P \cup \mathcal{M})}$  such that for all  $e \in E_s$ ,  $V(e) \subseteq (P_s \cup \mathcal{M})$  gives the label of the event  $e$ , that is, the propositions that hold after the execution of the event  $e$  and the messages that have been sent or received.

Given a Lamport diagram  $M = (E_{s_1}, \dots, E_{s_n}, \leq, V)$ , we can define the set of all configurations (global states) of  $M$  as  $\mathcal{C}_M \subseteq E_{s_1} \times \dots \times E_{s_n}$  such that every  $c = (e_1, \dots, e_n) \in \mathcal{C}_M$  satisfies the following consistency property:  $\forall i, j, \forall e \in E_j$  if  $e \leq e_i$  then  $e \leq e_j$ . Thus each configuration is a tuple of local states of services. Note that there is a unique initial global configuration  $(\perp_1, \dots, \perp_n) \in \mathcal{C}_M$ .

Let  $\alpha \in \Phi_s$  and  $e \in E_s$ . The notion that  $\alpha$  is true at the event  $e$  of service  $s$  in model  $M$  is denoted  $M, e \models_s \alpha$ , and is defined inductively as follows:

- $M, e \models_s p$  iff  $p \in V(e)$ .
- $M, e \models_s !a_{s'}$  iff  $\exists e' \in E_{s'}$  such that  $(e, e') \in <_c$  and  $a \in V(e')$ .
- $M, e \models_s ?a_{s'}$  iff  $\exists e' \in E_{s'}$  such that  $(e', e) \in <_c$  and  $a \in V(e')$ .
- $M, e \models_s \neg \alpha$  iff  $M, e \not\models_s \alpha$ .
- $M, e \models_s \alpha \vee \alpha'$  iff  $M, e \models_s \alpha$  or  $M, e \models_s \alpha'$ .
- $M, e \models_s \bigcirc \alpha$  iff there exists  $e' \in E_s$  such that  $e <_s e'$  and  $M, e' \models_s \alpha$ .
- $M, e \models_s \Diamond \alpha$  iff  $\exists e' \in E_s$ :  $e \leq_s e'$ ,  $M, e' \models_s \alpha$ .
- $M, e \models_s \ominus \alpha$  iff there exists  $e' \in E_s$  such that  $e' <_s e$  and  $M, e' \models_s \alpha$ .

When the send proposition  $!a_{s'}$  holds at  $e$  in service  $s$ , it means there is a corresponding receive event  $e'$  in service  $s'$  ( $(e, e') \in <_c$ ) and  $a$  holds locally in  $e'$ . Similarly, when the receive proposition  $?a_{s'}$  holds at  $e$  in service  $s$ , it means there is a corresponding send event  $e'$  in service  $s'$  ( $(e', e) \in <_c$ ) and  $a$  holds locally in  $e'$ .

Also, when  $\bigcirc \alpha$  holds at  $e$  in service  $s$ , it means that  $\alpha$  holds at  $e'$ , the one-step successor of  $e$ . Similarly, when  $\ominus \alpha$  holds at  $e$  in service  $s$ , it means that  $\alpha$  holds at  $e'$ , the one-step predecessor of  $e$ . Clearly, we see that  $M, e \models_s \ominus \text{False}$  iff  $e = \perp_s$ . Further, when  $\Diamond \alpha$  holds at  $e$  in service  $s$ , it means that  $\alpha$  holds at  $e'$ , a descendant of  $e$ .

For every global state  $c = (e_1, \dots, e_n) \in \mathcal{C}_M$  and global formula  $\psi \in \Psi$ , we define global satisfiability  $M, c \models \psi$  ( $\psi$  is true at configuration  $c$  of the model  $M$ ) inductively as follows:

- $M, c \models \alpha @ s_i$  iff  $M, e_i \models_{s_i} \alpha$ .
- $M, c \models \neg \psi$  iff  $M, c \not\models \psi$ .
- $M, c \models \psi_1 \vee \psi_2$  iff  $M, c \models \psi_1$  or  $M, c \models \psi_2$ .

Given a choreography  $\psi$  in  $p$ -LTL we define the set  $Models(\psi)$  as all the Lamport diagrams  $M$  such that  $M, c_0 \models \psi$ , where  $c_0$  is the unique initial global configuration of  $M$ .

### 3.2 Choreography Examples

The simplest choreography which can be encoded using  $p$ -LTL is that of producer-consumer protocol. This protocol describes the behaviour of two services, producer ( $p$ ) and consumer ( $c$ ), in which  $p$  produces objects labelled  $a$  which are consumed by  $c$ . The objects produced by  $p$  are put into a FIFO buffer



from where  $c$  retrieves them. Clearly, the protocol can be modelled as an asynchronous message passing system, where  $p$  sends messages labelled  $a$  to  $c$ . Depending on the size of buffer, there are various patterns of messages exchanged between  $p$  and  $c$ . Figure 6 gives the scenarios for the cases where buffer size is 1, 2 and 3.

The choreography for producer-consumer protocol may be formulated as  $\psi$  where  $\psi \stackrel{\text{def}}{=} \Box !a_c @ p \wedge \Box ?a_p @ c$ . It can be seen that Lamport diagrams in 6 are legitimate models of  $\psi$ .

Let us consider another choreography example. This concerns a system comprising three services: a traveller ( $T$ ) and map providers ( $M_1$  and  $M_2$ ). The GPS device of traveller ( $T$ ) has to automatically negotiate a purchase agreement with one of the two map providers. After  $T$  has already broadcast a “request of bid” message, the two services  $M_1$  and  $M_2$  send their respective bids.  $T$  evaluates the two bids and accepts one.

The message set is fixed as  $\mathcal{M} = \{bid, bid', acc, rej\}$ . We assume  $rej \equiv \neg acc$ . The choreography can be formulated as  $\psi \stackrel{\text{def}}{=} \alpha @ M_1 \wedge \beta @ M_2 \wedge \gamma @ T$  and:

- $\alpha \stackrel{\text{def}}{=} \Box (!bid_T \supset \Diamond (?acc_T \vee ?rej_T))$
- $\beta \stackrel{\text{def}}{=} \Box (!bid'_T \supset \Diamond (?acc_T \vee ?rej_T))$
- $\gamma \stackrel{\text{def}}{=} \Box ((?bid_{M_1} \supset \Diamond (!acc_{M_1} \vee !rej_{M_1})) \wedge (?bid_{M_2} \supset \Diamond (!acc_{M_2} \vee !rej_{M_2})) \wedge (\Diamond !acc_{M_1} \supset \Diamond !rej_{M_2}) \wedge (\Diamond !acc_{M_2} \supset \Diamond !rej_{M_1}))$

We briefly explain the local formulas:  $\alpha$  says that when a bid is send to  $T$  (by  $M_1$ ), it eventually receives either an acceptance or rejection.  $\beta$  says the same for  $M_2$ .  $\gamma$  says two things: when  $T$  receives a bid from  $M_1$  ( $M_2$ ) it either accepts or rejects it and, exactly one of the bids ( $bid$  or  $bid'$ ) is accepted.

The logical formalism which we have introduced in this section can not specify unrealizable choreographies of the kind mentioned in the previous section. Further, it is expected that software designers will not learn to write such formulas but will use tools that work with graphical formalisms and generate specifications interactively.

## 4 System of Communicating Automata

The service implementations for choreographies are given in terms of Systems of Communicating Automata (SCA). SCAs are quite similar to the automata model introduced in [21].

We fix  $n > 0$  and focus our attention on  $n$ -service systems. Let  $[n] = \{1, 2, \dots, n\}$ . A **distributed alphabet** for such systems is an  $n$ -tuple  $\tilde{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ , where for each  $i \in [n]$ ,  $\Sigma_i$  is a finite non-empty alphabet of actions of service  $i$  and for all  $i \neq j$ ,  $\Sigma_i \cap \Sigma_j = \emptyset$ . The alphabet induced by  $\tilde{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$  is given by  $\Sigma = \bigcup_i \Sigma_i$ . The set of **system actions** is the set  $\Sigma' = \{\lambda\} \cup \Sigma$ . The action symbol  $\lambda$  is referred

to as the **communication action**. This is used as an action representing a communication constraint through which every receive action will be dependent on its corresponding send action. We use  $a, b, c$  etc., to refer to elements of  $\Sigma$  and  $\tau, \tau'$  etc., to refer to those of  $\Sigma'$ .

**Definition 4.1.** A **System of  $n$  Communicating Automata (SCA)** on a distributed alphabet  $\tilde{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$  is a tuple  $S = ((Q_1, F_1), \dots, (Q_n, F_n), \rightarrow, \text{Init})$  where,

1. For each  $j \in [n]$ ,  $Q_j$  is a finite set of (local) states of service  $j$ .  
For  $j \neq j'$ ,  $Q_j \cap Q_{j'} = \emptyset$ .
2. for each  $j \in [n]$ ,  $F_j \subseteq Q_j$  is the set of (local) final states of service  $j$ .

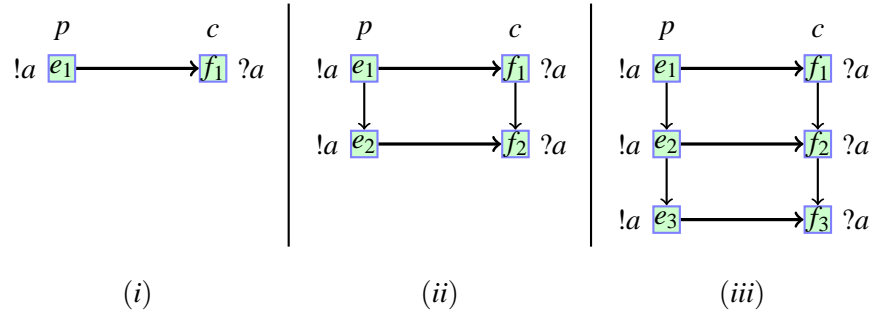


Figure 6: Lamport diagrams of the producer-consumer protocol

3. Let  $Q = \bigcup_j Q_j$ , then, the transition relation  $\rightarrow$  is defined over  $Q$  as follows.  $\rightarrow \subseteq (Q \times \Sigma' \times Q)$  such that if  $q \xrightarrow{\tau} q'$  then either there exists  $j$  such that  $\{q, q'\} \subseteq Q_j$  and  $\tau \in \Sigma_j$ , or there exist  $j \neq j'$  such that  $q \in Q_j, q' \in Q_{j'}$  and  $\tau = \lambda$ .
4.  $\text{Init} \subseteq (Q_1 \times \dots \times Q_n)$  is the set of global initial states of the system.

Thus, SCAs are systems of  $n$  finite state automata with  $\lambda$ -labelled communication constraints between them. The only ‘global’ specification is on initial states. This is in keeping with design of choreographies: the services are ‘set up’ and once initiated, manage themselves without global control.

Note that  $\rightarrow$  above is *not* a global transition relation, it consists of **local transition relations**, one for each service, and **communication constraints** of the form  $q \xrightarrow{\lambda} q'$ , where  $q$  and  $q'$  are states of different services. The latter define a coupling relation rather than a transition. The interpretation of local transition relations is standard: when the service  $i$  is in state  $q_1$  and reads input  $a \in \Sigma_i$ , it can move to a state  $q_2$  and be ready for the next input if  $(q_1, a, q_2) \in \rightarrow$ .

The interpretation of communication constraints is non-standard and depends only on automaton states, not on local input. When  $q \xrightarrow{\lambda} q'$ , where  $q \in Q_i$  and  $q' \in Q_j$ , it constrains the system behaviour as follows: whenever service  $i$  is in state  $q$ , it puts a message whose content is  $q$  and intended recipient is  $j$  into the buffer; whenever service  $j$  intends to enter state  $q'$ , it checks its environment to see if a message of the form  $q$  from  $i$  is available for it, and waits indefinitely otherwise. If a system  $S$  has no  $\lambda$  constraints at all, automata proceed asynchronously and do not wait for each other. We will refer to  $\lambda$ -constraints as ‘ $\lambda$ -transitions’ in the sequel for uniformity, but this explanation (that they are constraints not dependent on local input) should be kept in mind.

We use the notation  $\bullet q \stackrel{\text{def}}{=} \{q' \mid q' \xrightarrow{\lambda} q\}$  and  $q \bullet \stackrel{\text{def}}{=} \{q' \mid q \xrightarrow{\lambda} q'\}$ . For  $q \in Q$ , the set  $\bullet q$  refers to the set of all states from which  $q$  has incoming  $\lambda$ -transitions and the set  $q \bullet$  is the set of all states to which  $q$  has outgoing  $\lambda$ -transitions. The *global behaviour* of an SCA will be defined using its set of **global states**  $\tilde{Q} = Q_1 \times \dots \times Q_n$ . When  $\tilde{q} = (q_1, \dots, q_n) \in \tilde{Q}$ , we use the notation  $\tilde{q}[i]$  to refer to  $q_i$ . The language accepted by an SCA is a collection of ( $\Sigma$ -labelled) Lamport diagrams, to be defined below.

Figure 7 gives an SCA over the alphabet  $\tilde{\Sigma} = (\{!a\}, \{?a\})$ . The (global) initial state of this SCA is  $\{(q_0, q'_0)\}$  and the (global) final state is  $\{(q_2, q'_2)\}$ . The reader will observe that this SCA models the producer-consumer protocols given in Figure 6.

The producer generates the first object via the  $q_0 \xrightarrow{!a} q_1$  transition, any number of objects via the  $q_1 \xrightarrow{!a} q_1$  transition, and the last object via the  $q_1 \xrightarrow{!a} q_2$  transition. The consumer consumes the first object via the

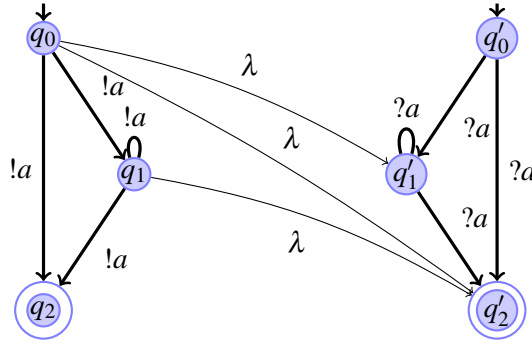


Figure 7: A simple SCA

$q'_0 \xrightarrow{?a} q'_1$  transition, any number of objects via the  $q'_1 \xrightarrow{?a} q'_1$  transition, and the last object via the  $q'_1 \xrightarrow{?a} q'_2$  transition. As a consumption can follow only after a production there is a  $\lambda$  transition between  $q_0$  and  $q'_1$  and  $q_0$  and  $q'_2$  and also between  $q_1$  and  $q'_2$ .

#### 4.1 Poset language of an SCA

We now formally define the run of an SCA on its input, a Lamport diagram and the poset language accepted by an SCA as the collection of Lamport diagrams on which the SCA has an accepting run.

Given an SCA  $S$  on  $\tilde{\Sigma}$ , a **run** of  $S$  on a Lamport diagram  $D = (E_1, \dots, E_n, \leq, V)$  is a map  $\rho : \mathcal{C}_D \rightarrow \tilde{Q}$ ,  $V : E \rightarrow \Sigma$ , such that the following conditions are satisfied:

- $\rho((\perp_1, \dots, \perp_n)) \in \text{Init}$ .
- For  $c \in \mathcal{C}_D$ , suppose  $\rho(c) = (q_1, q_2, \dots, q_n)$ . Consider  $c' \in \mathcal{C}_D$ , such that  $c$  differs from  $c'$  only at the  $i$ th position. Let  $e \in E_i$  be the  $i$ th element in  $c'$  and  $V(e) = \sigma \in \Sigma_i$ . Then,
  - $\rho(c') = (q'_1, q'_2, \dots, q'_n)$  where  $q'_j = q_j$  for all  $j \neq i$  and  $q_i \xrightarrow{\sigma} q'_i$  in  $S$ .
  - Suppose  $\exists e' \in E_j$ ,  $j \neq i$  such that  $e' <_c e$ . Let  $f' \in E'_j$  such that  $f' <_{c'} e'$ . Let  $c_0, c_1 \in \mathcal{C}'_D$  such that the  $j$ th element in  $c_0$  be  $f'$  and in  $c_1$  be  $e'$  whereas all the other elements are the same. Then,  $\rho(c_0)[j] \xrightarrow{V(e')} \rho(c_1)[j]$  and  $\rho(c_0)[j] \xrightarrow{\lambda} \rho(c')[i]$ . Remove  $\rho(c_0)[j]$  from the front of FIFO queue of  $j$  for  $i$ , if it is there else block.
  - If  $q_i \bullet \cap Q_j \neq \emptyset$ , then, there exists  $e' \in E_j$  such that  $e < e'$ . Insert  $q_i$  in the FIFO queue for  $j$ .

Thus, a run of  $S$  on  $D$  is a map from the set  $\mathcal{C}_D$  of configurations of  $D$  to the set of global states of  $S$  such that the following conditions hold: If  $c'$  is a configuration obtained by adding an event  $e \in E_i$  (where  $V(e) = \sigma$ ) to a configuration  $c$  then, there is a transition on  $\sigma$  from the local state of service  $i$  in  $\rho(c)$  to the local state of the same service in  $\rho(c')$  and all other local states are unaltered. In addition, if  $e$  is a receive event, we ensure that the corresponding send event has already occurred and that there is a  $\lambda$ -constraint into the resulting state. When there are out-going  $\lambda$ -constraints from the target state of the enabling transition, note that the definition makes sure that the corresponding event  $e$  is a send event and that it has a matching receive event.

Now, we specify the acceptance condition for a run  $\rho : \mathcal{C}_D \rightarrow \tilde{Q}$  of  $S$  over  $D$ . Let  $c = (e_1, \dots, e_n) \in \mathcal{C}_D$  such that for each  $i \in [n]$ ,  $e_i$  is the  $i$ -maximal event in  $D$ . The run  $\rho$  is said to be **accepting** if for

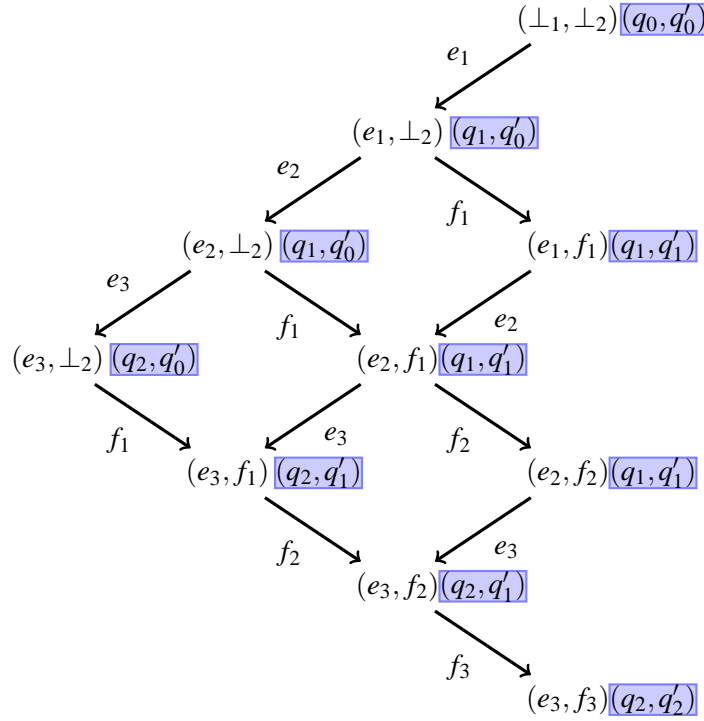


Figure 8: The run of SCA in 7 over Lamport diagram (iii) in 6

each  $i \in [n]$ ,  $\rho(c)[i] \in F_i$ . The **poset language accepted by  $S$**  is denoted by  $\mathcal{L}^{po}(S)$  and is defined as:  $\mathcal{L}^{po}(S) \stackrel{\text{def}}{=} \{D \mid D \text{ is a Lamport diagram and } S \text{ has an accepting run on } D\}$ .

For example, Figure 8 gives a run of the SCA in Figure 7 over the Lamport diagram (iii) of producer-consumer problem given in Figure 6. The figure essentially gives the directed acyclic graph corresponding to the configuration space of the Lamport diagram. Each node (configuration) has an associated state label given in shaded boxes on the right.

## 5 Realizability Algorithm

We now formulate the realizability problem for web service choreography in our setting and show that it is decidable. We do this by the so-called automata-theoretic approach of model checking. A composite web service implementation  $I$  is modelled as an SCA  $S$  and a choreography  $C$  is given by a formula  $\psi$  in  $p$ -LTL. Given a choreography  $\psi$ , the realizability problem is to check if there exists a composite web service implementation  $S$  that conforms to the choreography  $\psi$  i.e, to check if the global “behaviour” of  $S$  “satisfies”  $\psi$ . In order to do this, we give an algorithm to construct the system  $S_\psi$  accepting the models of  $\psi$ .

### 5.1 Formula Automaton for $p$ -LTL

In this section we show that one can effectively associate an SCA  $S_\psi$  with each  $p$ -LTL choreography  $\psi$  in such a way that the global behaviour of  $S_\psi$  satisfies the formula  $\psi$ : that is,  $\mathcal{L}^{po}(S_\psi) = \text{Models}(\psi)$ .

Given  $\psi$ , we first define a subformula closure set  $CL_s$  for each  $s \in Ag$ . This set  $CL_s$  of agent  $s$  is used to define the local states of that agent. Given a global formula  $\psi$ , the set  $CL(\psi)$  and  $CL_s$  for  $s \in Ag$ , are defined by simultaneous induction to be the least set of formulas such that:

1.  $\psi \in CL(\psi)$ .
2. if  $\alpha@s \in CL(\psi)$  then  $\alpha \in CL_s$ .
3.  $\{!a_{s'}, ?a_{s'}\} \subseteq CL_s$ , for each  $s' \in Ag, s \neq s'$ .
4.  $True \in CL_s$ ; we take  $\neg True$  as  $False$ .  $\bigcirc False \in CL_s$  and  $\ominus False \in CL_s$ .
5.  $\psi' \in CL(\psi)$  iff  $\neg\psi' \in CL(\psi)$ , taking  $\neg\neg\psi$  to be  $\psi$ .  $\alpha \in CL_s$  iff  $\neg\alpha \in CL_s$ , taking  $\neg\neg\alpha$  to be  $\alpha$ .
6. if  $\psi_1 \vee \psi_2 \in CL(\psi)$  then  $\psi_1, \psi_2 \in CL(\psi)$ . if  $\alpha_1 \vee \alpha_2 \in CL_s$  then  $\alpha_1, \alpha_2 \in CL_s$ .
7. if  $\bigcirc\alpha \in CL_s$  then  $\alpha \in CL_s$ .
8. if  $\diamond\alpha \in CL_s$  then  $\alpha, \bigcirc(\diamond\alpha) \in CL_s$ .
9. if  $\ominus\alpha \in CL_s$  then  $\alpha \in CL_s$ .

It can be checked that  $|CL(\psi)|$  and each  $|CL_s|$  are linear in the size of  $\psi$ . For the rest of this section, fix a global formula  $\psi_0 \in \Psi$ . We will refer to  $CL(\psi_0)$  simply as  $CL$  and  $CL_s$  will refer to the associated sets of  $s$ -local formulas. We also use  $U_s \stackrel{\text{def}}{=} \{\diamond\alpha \mid \diamond\alpha \in CL_s\}$ .

We say that  $A \subseteq CL_s$  is an  $s$ -atom iff it is locally consistent, that is, it contains  $True$  and:

1. for every formula  $\alpha \in CL_s$ , either  $\alpha \in A$  or  $\neg\alpha \in A$  but not both.
2. for every formula  $\alpha \vee \alpha' \in CL_s$ ,  $\alpha \vee \alpha' \in A$  iff  $\alpha \in A$  or  $\alpha' \in A$ .
3. for every formula  $\diamond\alpha \in CL_s$ ,  $\diamond\alpha \in A$  iff  $\alpha \in A$  or  $\bigcirc(\diamond\alpha) \in A$ .
4. if  $\ominus False \in A$  then for every  $\ominus\beta \in CL_s$ ,  $\ominus\beta \notin A$ .

An  $s$ -atom  $A$  is said to be *initial* if  $\ominus False \in A$

Let  $AT_s$  denote the set of all  $s$ -atoms. Let  $AT \stackrel{\text{def}}{=} \bigcup_s AT_s$ . Let  $\widetilde{AT}$  denote the set  $AT_1 \times \dots \times AT_n$ . We let  $\widetilde{A}, \widetilde{B}$  etc., to range over  $\widetilde{AT}$ , and  $\widetilde{A}[s]$  to denote the  $s$ -atom in the tuple.

Let  $\psi$  be a global formula. We define the notion  $\psi \in \widetilde{A}$  as follows:

1. for every  $s \in Ag$ , for every  $\alpha \in CL_s$ ,  $\alpha@s \in \widetilde{A}$  iff  $\alpha \in \widetilde{A}[s]$ ;
2. for every  $\neg\psi \in CL$ ,  $\neg\psi \in \widetilde{A}$  iff  $\psi \notin \widetilde{A}$ ;
3. for every  $\psi_1 \vee \psi_2 \in CL$ ,  $\psi_1 \vee \psi_2 \in \widetilde{A}$  iff  $\psi_1 \in \widetilde{A}$  or  $\psi_2 \in \widetilde{A}$ .

Given atoms  $A, A' \in AT_s$ , define the **local** relation  $\rightsquigarrow_\ell$  as follows:  $A \rightsquigarrow_\ell A'$  if and only if

1. for every  $\bigcirc\alpha \in CL_s$ ,  $\bigcirc\alpha \in A$  iff  $\alpha \in A'$ .
2. for every  $\ominus\alpha \in CL_s$ ,  $\ominus\alpha \in A'$  iff  $\alpha \in A$ .

The communication constraints are defined as follows: consider atoms  $A \in AT_s$  and  $B \in AT_{s'}$ ; define the **communication** relation  $\rightsquigarrow_\lambda$  as follows.  $A \rightsquigarrow_\lambda B$  if and only if

1. There exists  $A' \in AT_s$  such that  $A' \rightsquigarrow_\ell A$ ;
2. There exists  $B' \in AT_{s'}$  such that  $B' \rightsquigarrow_\ell B$ ;
3.  $!a_{s'} \in A'$ , and  $?a_s \in B$ .

We define local states for agent  $s$  as  $Q_s = AT_s \times U_s$ . We use  $\tilde{X}, \tilde{Y}$ , to represent members of  $\tilde{Q}$ , and  $\tilde{X}(A)[s], \tilde{X}(u)[s]$  etc., to denote the elements of the tuple in the  $s^{th}$  component.

We are now ready to associate an SCA with the given formula. For  $s \in Ag$ ,  $\Sigma_s \stackrel{\text{def}}{=} 2^{P_s \cup \mathcal{M}}$  constitute the distributed alphabet over which the SCA is defined.

**Definition 5.1.** *Given any formula  $\psi_0$ , the SCA associated with  $\psi_0$  is defined by:*

$$S_{\psi_0} \stackrel{\text{def}}{=} ((Q_{s_1}, F_{s_1}), \dots, (Q_{s_n}, F_{s_n}), \rightarrow, Init)$$

where:

1.  $Q_s = \{(A, u) \mid A \in AT_s, u \subseteq U_s\}$ .
2.  $F_s = \{(A, u) \in Q_s \mid \bigcirc False \in A, u = \emptyset\}$ .
3.  $Init = \{((A_1, \emptyset), \dots, (A_n, \emptyset)) \mid \psi_0 \in (A_1, \dots, A_n), \text{ and } A_s \text{ is initial for each } s\}$ .
4.  $(A, u) \xrightarrow{P'}_s (B, v)$ , where  $A, B \in AT_s$ , iff
  - (a)  $P' = \{p \in B \cap P_s\} \cup \{a \in \mathcal{M} \mid !a_{s'} \text{ or } ?a_{s'} \text{ is in } B \text{ for some } s' \in Ag\}$ .
  - (b)  $A \rightsquigarrow_\ell B$ .
  - (c) The set  $v$  is defined as follows:

$$v = \begin{cases} \{\diamond \alpha \in B \mid \alpha \notin B\} & \text{if } u = \emptyset \\ \{\diamond \alpha \in u \mid \alpha \notin B\} & \text{otherwise} \end{cases}$$

5.  $(A, u) \xrightarrow{\lambda}_s (B, v)$  iff  $A \rightsquigarrow_\lambda B$ .
6. For every  $s_i \neq s_j \in Ag$ , for every  $a \in \mathcal{M}$ , for every  $(A, u) \in Q_{s_i}$ , if  $!a_{s_j} \in A$  then there exists  $(B, v) \in Q_{s_j}$  such that  $(A, u) \xrightarrow{\lambda}_s (B, v)$ .

We denote  $S_{\psi_0}$  by  $S_0$  and assert the following with the proof in the appendix.

**Theorem 5.2.**  $Models(\psi_0) = \mathcal{L}^{po}(S_0)$ .

The above theorem says that, for a given choreography specification  $\psi_0$  in  $p$ -LTL, the set of service implementations  $S_0$ , constructed using the above algorithm, actually conform to the behaviour specified by  $\psi_0$ . This is so as every partial order execution of  $S_0$  is actually a model of the formula  $\psi_0$  and vice versa. Thus, every choreography expressed in  $p$ -LTL is realizable.

## 6 Implementation

The realizability algorithm for choreographies formulated in  $p$ -LTL, as given in the previous section, has been implemented. (The program is available from the authors on request.) Now, we briefly explain the program and discuss the experimental results.

This program is written in C and takes formulas as input, in text form. The input formula is preprocessed and converted to a tree form. First, we find the number and names of services from the input. We maintain two different arrays for positive and negative formulas in the closure sets of each service. The sizes of these sets are decided at run time and obtained from the size of input. We read the formula tree and identify subformulas pertaining to the services and put them in the closure set of the respective services. Extra formulas in the closure sets are generated and stored in tree form in the same arrays.

We consider the services one after another and generate atom sets for each and store them in a doubly linked list. A single atom in the NFA of a particular service  $s$  is interpreted as a boolean array of length  $|CL_s|$  and stored as a number between 0 and  $2^{|CL_s|} - 1$ . Similarly, the set of unfulfilled  $\diamond$ -requirements is taken as a boolean array of length  $|U_s|$  and stored as a number between 0 and  $2^{|U_s|} - 1$ . The states of  $s$  are another doubly linked list where each entry contains two integers, one from the atom set and another from the  $\diamond$ -requirement set. Thereafter, we take states from the state list two at a time and check whether the transition properties hold. If they do, the pair is put in the list for transitions, else dropped. This way, we generate transition set for each service  $s$  and store them in the respective list. Once, we have obtained all the transition sets (over all the services), we take two transitions from the lists of two different services and check whether they satisfy the properties pertaining to the coupling relation. If they do, we add the pair to the doubly linked list for the coupling relation else we drop it.

In the following table, we present some of the experimental results obtained from our implementation. The program is run on a laptop with 1.8 GiB RAM and a dual-core 2.10 GHz processor (Intel Pentium B950) and 32-bit OS (Ubuntu 12.04). We fix the number of services to 2 and input choreographies with different number of local modalities and send & receive propositions.

Size	Local	Send-Receive	States	Transitions	Couplings	Time (in ms)
7	2	0	16	32	0	10
9	4	0	64	128	0	14
5	0	2	8	16	16	< 1
5	1	2	80	320	8192	520
11	2	4	128	512	32768	1710

For different sizes of input choreographies, number of local temporal modalities ( $\bigcirc, \square, \diamond$  etc.), and number of send & receive propositions, we compute the number of states and transitions (over both the local automata), number of couplings ( $\lambda$ -transitions across the two automata) and the time taken (in ms) to generate the service implementations. Note that, when there are no send & receive propositions in the input, there are no couplings in the generated SCA. Further, as we increase the number of send & receive propositions by 2 with an attendant increase in local modalities by 1, the number of coupling shoots up by a multiple of 4. In fact, our suspicion is that too many useless (unreachable) states are being generated and consequently, the number of couplings is on a higher side.

## 7 Discussion

We have suggested that partial orders are a natural means for talking of web service interactions and proposed a decidable local logic for specifying global conditions on interacting web services such that every formula specifies a realizable choreography.

We have considered choreographies with finite partial order executions. A natural question relates to choreographies with infinite executions. The logic can be easily interpreted over such infinite behaviours as well, and extending SCAs to run on them is straightforward as well. We can thus show an analogue of Theorem 5.2, asserting realizability of specifications in the logic over infinite executions, but the technical details require some work.

The realizability algorithm which we have given in the paper is quite inefficient, in space as well as time, and needs improvement. Similarly the implementation needs to be fine tuned to make use of partial order methods and symmetries present in the global configuration space.

Another important area of further research would be improving the quality of the solution, that is, to move beyond realizability to realizing implementations with desired performance characteristics. For

instance, rather than specifying a fixed number of services *a priori*, we can ask for the minimal number of services realizing a choreography. The tradeoff between number of services and number of states of each service or communications between them can be relevant for applications.

A closely related question is when service formulas of  $p$ -LTL are used to specify service types instead of individual services. Multiple instances of services of each type may compose together consistently. Such a logic would clearly be more expressive, and its realizability problem is challenging.

An important theoretical question that arises from the discussion in the paper is the identification of the largest (satisfiable) subclass of LTL choreography properties that are realizable. While the paper presents a syntactic subclass that is sufficient, we need to expand it further.

While the paper discusses an initial theoretical investigation, what would be more interesting is the development of tools that facilitate analysis of specialized classes of choreographies, and we intend to pursue this.

## References

- [1] Bharat Adsul, Madhavan Mukund, K. Narayan Kumar & Vasumathi Narayanan (2005): *Causal Closure for MSC Languages*. In R. Ramanujam & Sandeep Sen, editors: *FSTTCS, Lecture Notes in Computer Science* 3821, Springer, pp. 335–347. Available at [http://dx.doi.org/10.1007/11590156\\_27](http://dx.doi.org/10.1007/11590156_27).
- [2] Rajeev Alur, Kousha Etessami & Mihalis Yannakakis (2005): *Realizability and verification of MSC graphs*. *Theor. Comput. Sci.* 331(1), pp. 97–114. Available at <http://dx.doi.org/10.1016/j.tcs.2004.09.034>.
- [3] Samik Basu, Tevfik Bultan & Meriem Ouederni (2012): *Deciding choreography realizability*. In: *POPL*, pp. 191–202. Available at <http://doi.acm.org/10.1145/2103656.2103680>.
- [4] Tevfik Bultan, Chris Ferguson & Xiang Fu (2009): *A Tool for Choreography Analysis Using Collaboration Diagrams*. In: *ICWS*, pp. 856–863. Available at <http://dx.doi.org/10.1109/ICWS.2009.100>.
- [5] Tevfik Bultan & Xiang Fu (2008): *Specification of realizable service conversations using collaboration diagrams*. *Service Oriented Computing and Applications* 2(1), pp. 27–39. Available at <http://dx.doi.org/10.1007/s11761-008-0022-7>.
- [6] Tevfik Bultan, Xiang Fu, Richard Hull & Jianwen Su (2003): *Conversation specification: a new approach to design and analysis of e-service composition*. In: *WWW*, pp. 403–410. Available at <http://doi.acm.org/10.1145/775152.775210>.
- [7] Tevfik Bultan, Xiang Fu & Jianwen Su (2007): *Analyzing Conversations: Realizability, Synchronizability, and Verification*. In: *Test and Analysis of Web Services*, pp. 57–85. Available at [http://dx.doi.org/10.1007/978-3-540-72912-9\\_3](http://dx.doi.org/10.1007/978-3-540-72912-9_3).
- [8] Marco Carbone, Kohei Honda & Nobuko Yoshida (2007): *Structured Communication-Centred Programming for Web Services*. In: *ESOP*, pp. 2–17. Available at [http://dx.doi.org/10.1007/978-3-540-71316-6\\_2](http://dx.doi.org/10.1007/978-3-540-71316-6_2).
- [9] Edmund M. Clarke, Orna Grumberg & Doron Peled (2000): *Model Checking*. MIT Press.
- [10] Gero Decker & Mathias Weske (2007): *Local Enforceability in Interaction Petri Nets*. In: *BPM*, pp. 305–319. Available at [http://dx.doi.org/10.1007/978-3-540-75183-0\\_22](http://dx.doi.org/10.1007/978-3-540-75183-0_22).
- [11] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen C. Hunt, James R. Larus & Steven Levi (2006): *Language support for fast and reliable message-based communication in singularity OS*. In: *EuroSys*, pp. 177–190. Available at <http://doi.acm.org/10.1145/1217935.1217953>.
- [12] X. Fu (2004): *Formal Specification and Verification of Asynchronously Communicating Web Services*. Ph.D. thesis, University of California, Santa Barbara.



- [13] Xiang Fu, Tevfik Bultan & Jianwen Su (2004): *Conversation protocols: a formalism for specification and verification of reactive electronic services*. *Theor. Comput. Sci.* 328(1-2), pp. 19–37. Available at <http://dx.doi.org/10.1016/j.tcs.2004.07.004>.
- [14] Sylvain Hallé & Tevfik Bultan (2010): *Realizability analysis for message-based interactions using shared-state projections*. In: *SIGSOFT FSE*, pp. 27–36. Available at <http://doi.acm.org/10.1145/1882291.1882298>.
- [15] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In: *POPL*, pp. 273–284. Available at <http://doi.acm.org/10.1145/1328438.1328472>.
- [16] Nickolas Kavantzaz, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon & Charlton Barreto (2005): *Web Services Choreography Description Language Version 1.0*. Technical Report, <http://www.w3.org/TR/ws-cdl-10/>.
- [17] Raman Kazhamiakin & Marco Pistore (2006): *Analysis of Realizability Conditions for Web Service Choreographies*. In: *FORTE*, pp. 61–76. Available at [http://dx.doi.org/10.1007/11888116\\_5](http://dx.doi.org/10.1007/11888116_5).
- [18] Niels Lohmann & Karsten Wolf (2009): *Realizability Is Controllability*. In: *WS-FM*, pp. 110–127. Available at [http://dx.doi.org/10.1007/978-3-642-14458-5\\_7](http://dx.doi.org/10.1007/978-3-642-14458-5_7).
- [19] Ashley T. McNeile (2010): *Protocol contracts with application to choreographed multiparty collaborations*. *Service Oriented Computing and Applications* 4(2), pp. 109–136. Available at <http://dx.doi.org/10.1007/s11761-010-0060-9>.
- [20] Ashley T. McNeile & Nicholas Simons (2006): *Protocol modelling: A modelling approach that supports reusable behavioural abstractions*. *Software and System Modeling* 5(1), pp. 91–107. Available at <http://dx.doi.org/10.1007/s10270-005-0100-7>.
- [21] B. Meenakshi & R. Ramanujam (2000): *Reasoning about Message Passing in Finite State Environments*. In: *ICALP*, pp. 487–498. Available at [http://dx.doi.org/10.1007/3-540-45022-X\\_41](http://dx.doi.org/10.1007/3-540-45022-X_41).
- [22] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS*, pp. 46–57. Available at <http://doi.ieeecomputersociety.org/10.1109/SFCS.1977.32>.
- [23] R. Ramanujam (1996): *Locally Linear Time Temporal Logic*. In: *LICS*, IEEE Computer Society, pp. 118–127. Available at <http://doi.ieeecomputersociety.org/10.1109/LICS.1996.561311>.
- [24] Ekkart Rudolph, Peter Graubmann & Jens Grabowski (1996): *Tutorial on Message Sequence Charts*. *Computer Networks and ISDN Systems* 28(12), pp. 1629–1641. Available at [http://dx.doi.org/10.1016/0169-7552\(95\)00122-0](http://dx.doi.org/10.1016/0169-7552(95)00122-0).
- [25] Gwen Salaün, Tevfik Bultan & Nima Roohi (2012): *Realizability of Choreographies Using Process Algebra Encodings*. *IEEE T. Services Computing* 5(3), pp. 290–304. Available at <http://doi.ieeecomputersociety.org/10.1109/TSC.2011.9>.
- [26] Jianwen Su, Tevfik Bultan, Xiang Fu & Xiangpeng Zhao (2007): *Towards a Theory of Web Service Choreographies*. In: *WS-FM*, pp. 1–16. Available at [http://dx.doi.org/10.1007/978-3-540-79230-7\\_1](http://dx.doi.org/10.1007/978-3-540-79230-7_1).
- [27] P. S. Thiagarajan (1995): *A Trace Consistent Subset of PTL*. In Insup Lee & Scott A. Smolka, editors: *CONCUR, Lecture Notes in Computer Science* 962, Springer, pp. 438–452. Available at [http://dx.doi.org/10.1007/3-540-60218-6\\_33](http://dx.doi.org/10.1007/3-540-60218-6_33).

## A Appendix: Proof of Correctness

We now prove theorem 5.2.

**Lemma A.1.**  $\mathcal{L}^{po}(S_0) \subseteq Models(\psi_0)$ .

*Proof.* Let  $D \in \mathcal{L}^{po}(S_0)$ . There is an accepting run  $\rho : \mathcal{C}_D \rightarrow \tilde{Q}$  of  $S_0$  on  $D$ . Let  $e$  be an  $s_i$ -event of  $D$ . We associate an  $s_i$ -atom  $A_e$  with  $e$  below. First let  $c$  be a configuration with  $e$  at the  $i$ -th position. Then,  $\rho(c)[i]$  is a tuple  $(A, u)$ , set  $A_e = A$  and similarly  $u_e = u$ .

We can define the valuation function for events of  $D$  as follows: for all  $e \in E_s$ ,  $V(e) \stackrel{\text{def}}{=} (A_e \cap P_s) \cup \{a \in \mathcal{M} \mid !a_{s_j} \in A_e\}$ .

The following assertion can be proved by induction on the structure of formulas in  $CL_s$ .

**Claim:** For all  $\alpha \in CL_s$ , for all  $e \in E_s$ ,  $D, e \models_s \alpha$  iff  $\alpha \in A_e$ .

Assuming the claim, we show that  $D \in Models(\psi_0)$ . Note that by construction of the automaton, if  $((A_1, u_1), \dots, (A_n, u_n)) \in Init$  then  $\psi_0 \in (A_1, \dots, A_n)$ .

Let  $c_0 = (\perp_1, \dots, \perp_n)$  be the initial global configuration.  $\rho(c_0) \in Init$ . Thus, we only need to show by above that for any global formula  $\psi$ , we have:  $D, c_0 \models \psi$  iff  $\psi \in (A_1, \dots, A_n)$ . This is shown by an easy induction.

- $(\psi_0 = \alpha@s)$ :  $\psi@s \in (A_1, \dots, A_n)$  iff  $\alpha \in A_{e_s}$  by the definition of membership in global atom  
iff  $D, \perp_s \models_s \alpha$  by Claim above  
iff  $D, c_0 \models \alpha@s$  by semantics.

The other cases are similar, by applying the induction hypothesis. Thus,  $D, c_0 \models \psi_0$  and hence  $D \models \psi_0$ . That is,  $D \in Models(\psi_0)$ , as required.

We proceed to prove the claim.

**Proof:**

$(\alpha = p)$   $D, e \models_s p$  iff  $p \in V(e)$  by definition of local satisfiability  
iff  $p \in A_e$  by the definition of  $V(e)$ .

$(\alpha = \bigcirc\beta)$  Suppose  $D, e \models_s \bigcirc\beta$ . We must show that  $\bigcirc\beta \in A_e$ . By the definition of  $\models_s$ , there exists  $e' \in E_s$  such that  $e \leq_s e'$  and  $D, e' \models_s \beta$ . Let  $c \in \mathcal{C}$  be a configuration with  $e$  as the  $s$ th element. Let  $c' \in \mathcal{C}$  be another configuration with  $e'$  as the  $s$ th element and all the other elements being same as that in  $c$ . By the definition of run, we have  $\rho(c)[s] \xrightarrow{A_{e'} \cap P_s} \rho(c')[s]$ . Therefore, for all  $\bigcirc\beta \in subf_s$ ,  $\bigcirc\beta \in A_e$  iff  $\beta \in A_{e'}$ .  $\therefore$  by the induction hypothesis,  $\beta \in A_{e'}$  hence, we have  $\bigcirc\beta \in A_e$  and we are done.

Conversely, suppose  $\bigcirc\beta \in A_e$ . We must show that  $D, e \models_s \bigcirc\beta$ . By the induction hypothesis and by the semantics of the modality  $\bigcirc$ , it suffices to prove that there exists  $e' \in E_s$  such that  $e \leq_s e'$  and  $\beta \in A_{e'}$ . Suppose not. Then,  $e$  must be the  $s$ -maximal event. So  $(A_e, u_e)$  must be a final state, but it is not. Therefore there exists  $e' \in E_s$  such that  $e \leq_s e'$ . Let  $c \in \mathcal{C}$  be a configuration with  $e$  as the  $s$ th element. Let  $c' \in \mathcal{C}$  be another configuration with  $e'$  as the  $s$ th element and all the other elements being same as that in  $c$ . By the definition of run, we have  $\rho(c)[s] \xrightarrow{A_{e'} \cap P_s} \rho(c')[s]$ . Therefore, for all  $\bigcirc\beta \in subf_s$ ,  $\bigcirc\beta \in A_e$  iff  $\beta \in A_{e'}$ . Now,  $\bigcirc\beta \in A_e$  as given so  $\beta \in A_{e'}$ . Thus, we are done.

$(\alpha = \Diamond\beta)$  Suppose  $D, e \models_s \Diamond\beta$ . We must show that  $\Diamond\beta \in A_e$ . Since  $D, e \models_s \Diamond\beta$ , there exists  $e' \in E_s$  such that  $e \leq_s e'$ ,  $D, e' \models_s \beta$ . That is, by induction hypothesis, there exists  $e' \in E_s$  such that  $e \leq_s e'$ ,  $\beta \in A_{e'}$ . We need to show that  $\Diamond\beta \in A_e$ .

Let  $e = e_1 \leq_s e_2 \leq_s \dots \leq_s e_k = e'$  be the sequence of events through which  $e'$  is reached from  $e$ . We show that  $\Diamond\beta \in A_e$  by a second induction on  $l = k - 1$ .

*Base case:* ( $l = 0$ ).

Then,  $k = 1$  and so  $D, e \models_s \beta$ . By the main induction hypothesis,  $\beta \in A_e$  and (by the definition of atom),  $\Diamond\beta \in A_e$ .

*Induction step:* ( $l > 0$ ).

By the semantics of the modality  $\Diamond$ ,  $D, e \models_s \neg\beta$  and  $D, e_2 \models_s \Diamond\beta$ . Therefore, by the secondary induction hypothesis,  $\Diamond\beta \in A_{e_2}$ . From the definition of  $\rightarrow$ , we have  $\bigcirc(\Diamond\beta) \in A_e$ . By the main induction hypothesis, we have  $\neg\beta \in A_e$  as well. Combining these facts and using the definition of an atom, we see that  $\Diamond\beta \in A_e$  as required.

Conversely, suppose  $\Diamond\beta \in A_e$ . We must show that  $D, e \models_s \Diamond\beta$ . Since  $\rho$  is an accepting run of  $S_0$ , there is a maximal event  $e' \in E_s$ . Now suppose that  $\neg\beta \in A_{e''}$  for every  $e \leq e'' \leq e'$ . Then by an argument similar to the above, we can show that  $\bigcirc\Diamond\beta \in A_{e'}$  for every  $e \leq e'' \leq e'$ . Thus we get  $\bigcirc\Diamond\beta \in A_{e'}$  at the maximal event  $e'$  contradicting the fact that  $\bigcirc\text{False} \in A_{e'}$ . Thus, there exists  $e''$  such that  $e \leq e'' \leq e'$  and  $\beta \in A_{e''}$ . Then what we need follows by induction hypothesis.

$(\alpha = \ominus\beta) \Rightarrow$  Given  $D, e \models_s \ominus\beta$ . By the definition of local satisfiability, there exists  $e' \in E_s$  such that  $e' \leq_s e$  and  $D, e' \models \beta$ . Let  $c \in \mathcal{C}$  be a configuration with  $e$  as the  $s$ th element. Let  $c' \in \mathcal{C}$  be another configuration with  $e'$  as the  $s$ th element and all the other elements being same as that in  $c$ . By the definition of run, we have  $\rho(c')[s] \xrightarrow{A_e \cap P_s} \rho(c)[s]$ . By the definition of  $\rightarrow_s$ ,  $\ominus\beta \in A_e$  iff  $\beta \in A_{e'}$ . By induction hypothesis,  $\beta \in A_{e'}$ , so  $\ominus\beta \in A_e$  and we are done.

$(\Leftarrow)$  Given  $\ominus\beta \in A_e$ . It suffices to show that there exists  $e' \in E_s$  such that  $e' \leq_s e$  and  $\beta \in A_{e'}$ . Suppose there is no  $e'$  such that  $e' \leq_c e$ . That is,  $e$  is the  $s$ -minimum event. Then,  $(A_e, u_e) \in I_s$ . So, for every  $\ominus\gamma \in CL_s$ ,  $\ominus\gamma \notin A_e$ . This is a contradiction. Therefore, there exists  $e' \in E_s$  such that  $e' \leq_s e$ . Now, let  $c \in \mathcal{C}$  be a configuration with  $e$  as the  $s$ th element. Also, let  $c' \in \mathcal{C}$  be another configuration with  $e'$  as the  $s$ th element and all the other elements being same as that in  $c$ . By the definition of run, we have  $\rho(c')[s] \xrightarrow{A_e \cap P_s} \rho(c)[s]$ . By the definition of  $\rightarrow_s$ ,  $\ominus\beta \in A_e$  iff  $\beta \in A_{e'}$ . Therefore,  $\beta \in A_{e'}$  as we already have  $\ominus\beta \in A_e$ . Thus, we have shown that there exists  $e' \in E_s$  such that  $e' \leq_s e$  and  $\beta \in A_{e'}$  and we are done.

$(\alpha = !a_{s_j}) \Rightarrow$  Given  $D, e \models_s !a_{s_j}$ . There exists  $e' \in E_{s_j}$  such that  $e <_c e'$  and  $a \in V(e)$ . By the definition of run,  $(A_e, u_e) \xrightarrow{\lambda} (A_{e'}, u_{e'})$ . By the definition of  $\rightsquigarrow_\lambda$ ,  $!a_{s_j} \in A_e$ .

$(\Leftarrow)$  Given  $!a_{s_j} \in A_e$ . There exists  $(A_{e'}, u_{e'}) \in Q_{s_j}$  such that  $(A_e, u_e) \xrightarrow{\lambda} (A_{e'}, u_{e'})$ . By the definition of the run  $e <_c e'$ . By the definition of  $V$ ,  $a \in V(e)$ . Therefore,  $D, e \models_s !a_{s_j}$ .

$(\alpha = ?a_{s_j})$  The reasoning about  $?a_{s_j}$  is similar to that of  $!a_{s_j}$  as given above.

□

**Lemma A.2.**  $Models(\psi_0) \subseteq \mathcal{L}^{p_0}(S_0)$ .

*Proof.* Conversely, suppose  $D \models \psi_0$ , where  $D = (E_{s_1}, \dots, E_{s_n}, \leq_{s_1}, \dots, \leq_{s_n}, <_c V)$ . To show that  $D$  is a member of  $\mathcal{L}^{p_0}(S_0)$ , we have to construct an accepting run of  $\mathcal{L}^{p_0}(S_0)$  on  $D$ .

For every  $s \in Ag$ , for every  $e \in E_s$ , define the set  $A_e$  as follows:

$$A_e = \{\alpha \in CL_s \mid D, e \models \alpha\}$$

Let  $e_s^0$  be the minimum event in  $E_s$ . We construct  $A_{\perp_s}$  from  $A_{e_s^0}$  as follows:

$$A_{\perp_s} = \Delta A_{\perp_s} \cup \{\neg\alpha \in CL_s \mid \alpha \notin \Delta A_{\perp_s}\} \cup \{\alpha \vee \beta \in CL_s \mid \alpha \in \Delta A_{\perp_s}\} \cup \{\bigcirc\Diamond\alpha \mid \Diamond\alpha, \neg\alpha \in \Delta A_{\perp_s}\} \quad \text{where}$$

$$\Delta A_{\perp_s} = \{\neg p \mid p \in CL_s \cap P_s\} \cup \{\neg!a_{s'}, \neg?a_{s'} \mid !a_{s'}, ?a_{s'} \in CL_s\} \cup \{\neg\ominus\beta \mid \ominus\beta \in CL_s\} \cup \delta A_{e_s^0} \quad \text{and}$$

$$\delta A_{e_s^0} = \{\bigcirc\alpha \in CL_s \mid \alpha \in A_{e_s^0}\} \cup \{\alpha \mid \ominus\alpha \in A_{e_s^0}\} \cup \{\ominus False\}$$

For every  $s \in Ag$ , for every  $f \in E'_s$ , define the set  $u_f$  inductively as follows:

- $u_{\perp_s} = \emptyset$ ,
- for every  $f, f' \in E'_s$  such that  $f <_s f'$ ,

$$u_{f'} = \begin{cases} \{\Diamond\alpha \in A_{f'} \mid \alpha \notin A_{f'}\} & \text{if } u_e = \emptyset \\ \{\Diamond\alpha \in u_f \mid \alpha \notin A_{f'}\} & \text{otherwise} \end{cases}$$

Now, for any configuration  $c = (f_{s_1}, \dots, f_{s_n})$  in  $\mathcal{C}_D$  define

$$\rho(c) = \langle (A_{f_{s_1}}, u_{f_{s_1}}), \dots, (A_{f_{s_n}}, u_{f_{s_n}}) \rangle.$$

It is now easily shown that  $\rho$  is an accepting run of  $\mathcal{L}^{po}(S_0)$  on  $D$  and hence,  $D \in \mathcal{L}^{po}(S_0)$  and we are done.  $\square$

The two foregoing lemmata A.1 and A.2, together, give us the theorem 5.2.